WebAPI: A Native Contract Standard for Web Components

Amu Hlongwane
Software Engineering Labs Pty Ltd (SWE Labs)
amu.hlongwane@swelabs.io

Elvis Hlongwane
Software Engineering Labs Pty Ltd (SWE Labs)
engineers@swelabs.io

Abstract

Backend APIs have OpenAPI: a vendor-neutral, machine-readable, enforceable contract format covering an endpoint's request and response shape, adopted across languages and frameworks under independent governance. No equivalent exists for a frontend UI component. HTML, CSS, the DOM, and Web Components are construction primitives rather than a contract layer; existing partial standards — WAI-ARIA, the Design Tokens Community Group format, Custom Elements Manifest — each cover one dimension without unifying props, structure, events, state, accessibility, and style into one enforceable contract. This paper argues the gap is real, evidenced by independent engineering efforts that separately converged on solving it, including at least one public, design-tool-coupled project and at least one native, framework-independent implementation (documented separately [9]). Comparing a design-tool-coupled candidate against the requirements of a framework-independent contract shows the two are not interchangeable: a schema built around one design tool and one component framework encodes structural assumptions — required design-tool bindings, framework-specific handler naming, module-export-shaped anchors — that a native Web Component cannot satisfy. We propose WebAPI: a contract model independent of any specific UI framework or design tool, occupying the same relationship to frontend frameworks that OpenAPI occupies to backend frameworks. We define the six dimensions such a contract must represent, identify the tooling gaps still open even where a native implementation already demonstrates feasibility, and propose a governance path modeled on OpenAPI's own pre-standardization history rather than a W3C or WHATWG submission, which we argue is the wrong venue for a component-authoring format.

Keywords

web components, API contracts, OpenAPI, UI standards, design systems, software architecture

Type: Technical white paper, submitted to the ICSE NIER (New Ideas and Emerging Results) track. Date: August 2026.

1 Problem Statement

1.1 The Backend Has a Contract Standard

A REST API's shape [11] — its endpoints, request and response schemas, status codes, and authentication requirements — is described by OpenAPI [1], a specification with no single owner, no proprietary tier, and a large surrounding tooling ecosystem: validators, client and server code generators, and interactive documentation viewers. A backend team can declare its API's contract once and have that contract validated, documented, and consumed by unrelated tools without further coordination.

1.2 The Frontend Has No Equivalent

A UI component's full contract — its typed inputs, the events it emits, the states it can occupy, its accessibility semantics, and the design tokens it consumes — has no equivalent standard. What exists natively in the browser platform addresses only fragments of this:

A common but incorrect intuition is that Web IDL [4] — the language browser specifications use to define native-to-JavaScript interfaces — fills this role. It does not. Web IDL describes the boundary between a browser engine's native implementation and the JavaScript objects it exposes (`fetch()`, `Element`, `localStorage`), a concern orthogonal to describing an *application's own* component contract. Most Web IDL-defined interfaces (Canvas, Geolocation, the DOM itself [3]) have no relationship to backend communication or component authoring at all, which rules out treating Web IDL as an implicit answer to this gap.

1.3 Research Questions

1. Is the absence of a frontend contract standard a genuine, unmet need, or a solved problem that has simply not been named?
2. What would a complete contract need to represent, independent of any one UI framework or design tool?
3. What governance path is appropriate for a format of this kind?

---

2 Related Work

2.1 Partial, Native Standards

Web Components [2], WAI-ARIA [5], the DTCG token format [6], and Custom Elements Manifest [7] each formalize one slice of a component's surface — construction, accessibility semantics, style values, and descriptive metadata respectively — without a governing format that unifies them into one enforceable contract, as summarized in Table 1.

2.2 Design-Tool-Coupled Contract Formats

At least one public, independently governed project [8] addresses this gap directly by coupling a component contract to a specific UI framework and a specific design tool. Its contract format covers typed props, structural anatomy, design tokens, a fixed set of interaction states, accessibility semantics, and declared events, generating both a framework implementation and a design-tool library from one source contract, reconciled by a working provenance mechanism that tracks which of its two surfaces last changed. Section 4 examines why this coupling, while a real strength for its intended use case, makes the format unsuitable as a framework-independent standard unless its structure changes.

Table 1: Existing standards addressing fragments of a UI component's contract.

Standard	Scope	Gap
Web Components (Custom Elements, Shadow DOM)	Encapsulated construction primitive	No contract or validation layer
WAI-ARIA	Accessibility semantics	Accessibility only
Design Tokens Community Group (DTCG) format	Token value representation	Values only, no component structure
Custom Elements Manifest	Descriptive metadata for a custom element's API	Documentation only, no enforcement

2.3 Native, Framework-Independent Implementations

A separate line of work demonstrates that a framework-independent contract meeting the requirements set out in Section 5 is implementable in a native Web Component toolchain today, including a companion runtime addressing mechanism for automation and agent consumption. That implementation, and the addressing mechanism specifically, are documented in a dedicated paper [9] and are treated here as an existence proof for feasibility rather than re-derived in full.

2.4 Runtime Agent Exposure

WebMCP [10] — a draft specification under the W3C Web Machine Learning Community Group, jointly developed by the Chrome and Edge teams — proposes a `navigator.modelContext` interface through which a page declares discoverable, callable tools to an in-browser AI agent. This addresses a different boundary than the one this paper is concerned with: it standardizes how a page's *runtime* capabilities are exposed to an agent, not how a component's *build-time* contract is declared and validated. Section 6 discusses it as a candidate downstream consumer of a WebAPI contract, not as an alternative to one.

3 Evidence the Gap Is Real

The strongest evidence for an unmet need is not the absence of a standard but the existence of independent, unrelated attempts to fill it. At least two such attempts were identified: a design-tool-coupled public project (Section 2.2) and a native, framework-independent implementation (Section 2.3), developed without reference to one another. Two serious, independent efforts converging on the same *category* of problem while diverging sharply on *shape* — one requiring a specific design tool and framework, the other requiring neither — is consistent with a pre-standardization state, not a settled one. If the problem had an obvious, agreed shape, independent implementations would be expected to converge on similar fields, the way REST API implementations converge on a broadly similar shape once a schema format such as OpenAPI exists to describe them.

4 The Case for a Native, Framework-Independent Standard

The design-tool-coupled candidate examined in Section 2.2 is a mature, working system, but its schema encodes structural commitments to one framework and one design tool rather than remaining framework-independent the way OpenAPI is independent of any specific backend language:

- Every prop's binding declaration requires both a design-tool binding and a code binding; the schema provides no valid way to omit either.
- Every event's code binding is constrained to a single framework's handler-naming convention (a `on`-prefixed, capitalized property name). A native Web Component does not have handler props at all — it emits a `CustomEvent` consumed through `addEventListener`. The field is therefore not missing an optional value; it has no valid value for a native component.
- Code bindings anchor to a module import/export pair — which exported component a contract binds to in that framework's module system. A custom element binds by tag name and runtime registration, not by module export.

These are architectural commitments, not gaps that additional optional fields would close. Adopting this schema for a native, framework-independent toolchain would require either populating required fields with no honest value, or forking the schema into a variant no longer compatible with its originating project's own tooling — which defeats the purpose of adopting a shared format at all.

This motivates the paper's central claim: the correct target is a contract standard independent of any one UI framework or design tool, occupying the same relationship to frontend frameworks that OpenAPI occupies to backend frameworks — consumable by any framework built on Web Components, because Web Components are the one construction primitive already common to all of them, rather than privileging one framework and one design tool by construction.

5 Proposed Contract Model: WebAPI

Table 2 restates the OpenAPI-parity requirement identified at the outset: each dimension a REST contract requires has a UI-contract equivalent that must be representable for the analogy to hold. We refer to a contract meeting all six as a **WebAPI contract**.

Table 2: OpenAPI concept mapped to its required WebAPI equivalent.

OpenAPI concept	WebAPI equivalent
Request schema	Typed props (required/optional, enums, defaults)
Response schema	Structural contract (elements, slots, cardinality)
Status codes	Component states
Headers	Events (name and payload schema)
Security schemes	Accessibility contract
Servers	Design tokens consumed
Path versioning	Contract version and breaking-change semantics
Validators	Build-time schema validation
Generated client SDKs	Generated component bindings
Interactive documentation (e.g., Swagger UI)	A contract viewer
Contract diff tooling	Breaking-change detection between versions
Independent governing body	Neutral, cross-vendor governance

A WebAPI contract, by this definition, is deliberately silent on which UI framework consumes it and which design tool, if any, a team also maintains alongside it. Framework and design-tool bindings — where a team wants them — are additive layers on top of a WebAPI contract, not part of the contract's required shape, mirroring how OpenAPI itself is silent on which backend language implements an endpoint.

6 Feasibility and Remaining Gaps

6.1 Feasibility

A specification with no working implementation is a proposal, not evidence of feasibility. The native, framework-independent implementation cited in Section 2.3 demonstrates that every structural dimension in Table 2 — excluding the final three, addressed below — is representable in a format independent of any specific UI framework or design tool: typed props with composition support, named elements and slots with cardinality, events with typed payloads, a state model, markup-level accessibility semantics, and design-token references, together with a companion addressing mechanism for runtime and automation consumers. Readers seeking the field-level detail of that implementation should consult its own documentation [9] rather than this paper, which treats it as an existence proof rather than a specification of record.

This is specifically a **schema-representability** claim, not an empirical one: the fields exist in a real, working schema and were confirmed by direct inspection of that schema. It is a distinct and weaker claim than "the model performs well in production," which the cited implementation's own paper does not yet establish — its empirical evaluation is specified as a protocol but not yet executed [9]. The feasibility argument here therefore rests on what the schema can express, not on measured outcomes from using it.

6.2 Remaining Gaps

Three gaps remain even in the most complete implementation surveyed, none of which are contract-model deficiencies:

No contract viewer. Backend tooling commonly generates an interactive explorer for an OpenAPI document. No equivalent exists for browsing a WebAPI contract's props, events, states, accessibility semantics, or design tokens.

No enforced breaking-change detection. A contract version field's intended purpose — flagging a breaking change to props, structure, or address contracts — requires tooling that checks it; documenting the intent is not the same as enforcing it. The design-tool-coupled candidate in Section 2.2 offers a working, content-hash-based provenance mechanism as one proven reference design for this problem, independent of any decision about adopting the rest of its schema.

No runtime agent-exposure bridge. Neither implementation surveyed exposes a component's interactive affordances to an in-browser AI agent through a standards-track mechanism. WebMCP (Section 2.4) is offered as a candidate downstream target: a component's modeled interactive elements, valid actions, and typed events are close to what such a manifest requires, but we scope this deliberately as an *additional*, optional consumption target generated from an existing contract, not a requirement for the contract itself to be considered complete — the same way OpenAPI does not require a specific transport binding to be a complete contract.

7 Governance Path

OpenAPI is not a W3C or WHATWG standard. It began as a single company's format (Swagger) and reached its current status through independent, neutral governance (the OpenAPI Initiative, under the Linux Foundation) and cross-vendor adoption, not through submission to a browser standards body. WebMCP is the one component adjacent to this proposal that *does* belong on a browser-standards track, because it is a runtime API a browser itself must implement — but it is an existing proposal from other organizations, not one this paper originates, and adopting it as a downstream target is a contribution decision, not an authoring one.

The recommended path for the WebAPI contract format mirrors OpenAPI's own history rather than a standards-body submission:

1. Publish the WebAPI contract model as an independently versioned specification, decoupled from any single compiler's implementation, the way OpenAPI's specification is decoupled from any one code generator.

2. Build and publish the missing tooling identified in Section 6.2 — a contract viewer and breaking-change enforcement — as reference tooling against the independent specification.
3. Seek adoption from at least one framework or toolchain outside the originating organization before pursuing neutral, cross-vendor governance; governance follows adoption rather than preceding it.

---

8 Threats to Validity

This is a conceptual paper; its claims rest on direct source and specification inspection rather than a controlled study, and the corresponding limitations are stated plainly below.

Feasibility evidence drawn from a separately documented implementation. Section 6.1's feasibility claim rests on a native implementation documented in a companion paper [9] rather than re-derived here. No claim is made that any specific existing implementation would be adopted as-is; it demonstrates feasibility, not readiness for external use.

Comparison snapshot, not longitudinal tracking. The design-tool-coupled candidate discussed in Sections 2.2 and 4 reflects a reading of its schema at one point in time. Actively developed projects change; the comparison should be re-verified before being treated as current in any future revision of this work.

No formal usability or adoption study. No developer study, survey, or controlled comparison was conducted to assess whether practitioners would prefer a framework-independent WebAPI contract over ad hoc conventions or a design-tool-coupled approach. The argument in this paper is architectural and evidentiary — what each schema can and cannot represent — not empirical in the sense of measuring developer outcomes.

---

9 Conclusion

Backend engineering has treated a contract standard for its API surface as necessary for over a decade; frontend engineering has not yet produced an equivalent for its component surface, despite the same underlying need — a machine-readable, enforceable description of what a unit of software promises to its consumers. This paper found that need to be real, evidenced by independent, structurally divergent attempts to address it, and proposed WebAPI: a contract model independent of any specific UI framework or design tool, defined across six dimensions that parallel the requirements a REST contract already meets. A separately documented native implementation demonstrates the model is buildable today, with specific, named tooling gaps rather than an unbounded one. The appropriate next step is not a browser-standards submission but the path OpenAPI itself took: an independent specification, proven tooling, and adoption sought before governance is sought.

---

10 References

1. OpenAPI Initiative. "OpenAPI Specification." [Online]. Available: https://www.openapis.org/. [Accessed: Aug. 2026].
2. WHATWG. "HTML Living Standard." [Online]. Available: https://html.spec.whatwg.org/. [Accessed: Aug. 2026].
3. WHATWG. "DOM Living Standard." [Online]. Available: https://dom.spec.whatwg.org/. [Accessed: Aug. 2026].
4. W3C. "Web Interface Definition Language (Web IDL)." [Online]. Available: https://webidl.spec.whatwg.org/. [Accessed: Aug. 2026].
5. W3C. "Accessible Rich Internet Applications (WAI-ARIA)." [Online]. Available: https://www.w3.org/TR/wai-aria/. [Accessed: Aug. 2026].
6. W3C Design Tokens Community Group. "Design Tokens Format Module." [Online]. Available: https://www.w3.org/community/design-tokens/. [Accessed: Aug. 2026].
7. open-wc. "Custom Elements Manifest." [Online]. Available: https://github.com/open-wc/custom-elements-manifest. [Accessed: Aug. 2026].
8. southleft. "Design System Contracts." [Online]. Available: https://github.com/southleft/ds-contracts-poc. [Accessed: Aug. 2026].
9. Hlongwane, A. (Software Engineering Labs), with contributing organizations PHD Group and Agentic Labs. "Deterministic DOM Addressing System: A Contract Model for AI-Native Web Interaction." Position/vision paper, March 2026. [Online]. Available: https://github.com/sweengineeringlabs/paperlint/blob/dev/docs/papers/dom_address_system_research.md (dev branch). — Companion paper documenting the native, framework-independent reference implementation (DOMAddress/DDAS) and its runtime addressing mechanism cited in Sections 2.3 and 6.1. Its formal model (ID derivation, breaking-change semantics, schema) is specified and implemented; its empirical evaluation (production-application stability study, agent-interaction study) is proposed as a protocol but not yet executed — see Section 6.1 for what this paper's feasibility claim does and does not rely on as a result.
10. W3C Web Machine Learning Community Group. "WebMCP." [Online]. Available: https://w3c-cg.github.io/aikr/webMCP/webmcp-technical-notes.html. [Accessed: Aug. 2026].
11. Fielding, R. T. (2000). "Architectural Styles and the Design of Network-based Software Architectures." PhD dissertation, University of California, Irvine. — On REST as an architectural style rather than a formal protocol.

10.1 Future Work

- **Standalone specification publication.** Publish the WebAPI contract model (Section 5) as an independently versioned, implementation-agnostic specification, as proposed in Section 7.
- **Contract viewer and breaking-change tooling.** Implement the two concrete tooling gaps identified in Section 6.2, with the latter modeled on the provenance mechanism referenced in Section 2.2.
- **WebMCP export prototype.** Generate a `navigator.modelContext` tool manifest from an existing WebAPI contract's interactive elements and events, independent of the broader specification-publication effort.
- **External adoption study.** Once an independent specification exists, assess adoption willingness among at least one framework or toolchain team outside the originating organization, as a precondition for pursuing neutral governance per Section 7.